Miami University

Department of Electrical and Computer Engineering

-Convolutional Neural Network-

-SmartNet Lab Report-

Build a CNN model to for image classification from CIFAR-10
with Tensorflow library

Sunday, February 27, 2022

Written By:
-Loc Tran-

Date: March 1, 2022

Contents

1 Introduction

In this assignment, I am going to build a Convolutional Neural Network model from Tensorflow library to predict the label in each image from **CIFAR-10**. The CIFAR-10 data consists of 60,000 (32×32) color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images in the official data. The label classes in the dataset are:

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- house
- ship
- truck

I will apply a Sequential Neural Network architecture which consists of approximately 6 *Convolutional 2D layers*, along with some *Max Pooling* layers and *Drop Out* regularization method to get a result: classifying 10,000 images into 10 classes.

2 Pre-processing

To create a model, I need to prepare the dataset for it so that the I can classify them. In this model, I will not use any of data from my own. Instead, as I have said, I imported CIFAR-10 from keras library using this line of code:

```
from tensorflow.keras import datasets
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
```

Then, I have had my data set. These images below are some examples from the set:

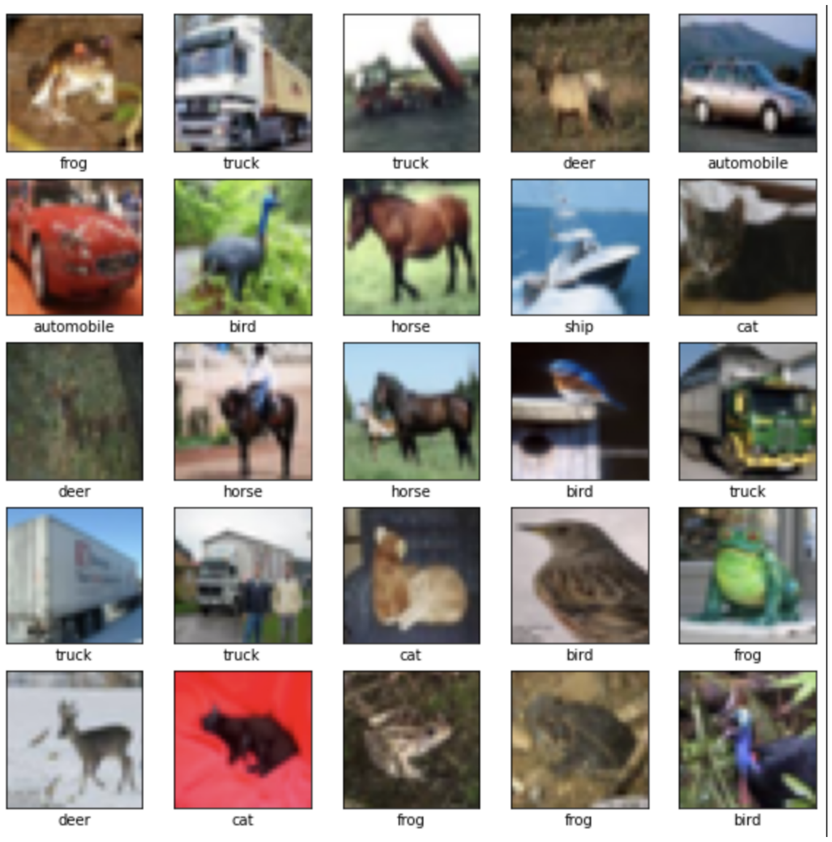


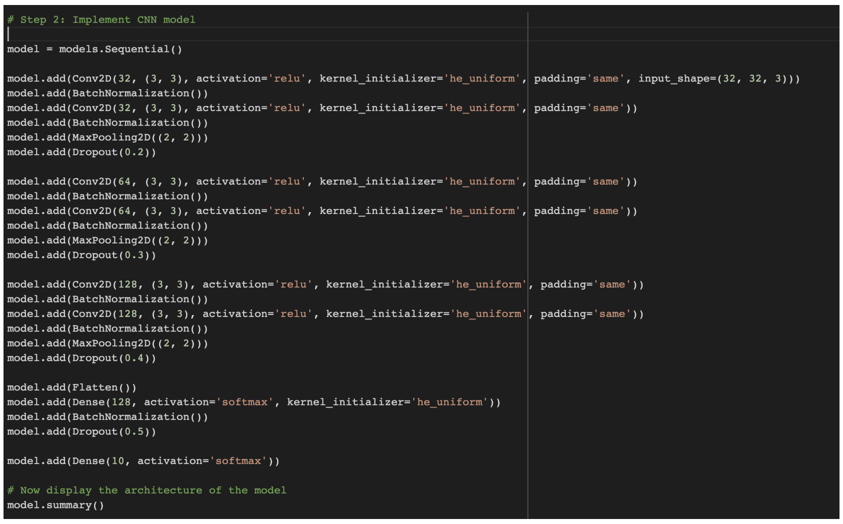
Figure 2.1: Data set

3 Implement Model

This is the most important part of the project, when I needed to build a model based on those given images so that it can make further prediction. To build this model, I used **Tensorflow** library as an consistent method to make the prediction with a few lines of code, which help developers design and maintain the project easier. My model was built under Neural Network structure (like human brain) in which each layer will receive inputs from previous, plug them in activation function, calculate the loss function, and return to backward-propagation to find gradient descent, which can optimize the weight(W) and bias(b) parameters.

However, I had to do some few more steps because this dataset is extremely big and mode complicated then the previous model. I had to add 6 layers, which are Convolutional 2-Dimensional layers along with **he uniform** as parameters initializer and **same** as padding. Moreover, each layer is followed by a **Batch Normalization** layer to avoid overfitting for batch optimization, and after every 2 Convolutional layers with Batch Normalization, I also add a **Max Pooling** 2D layer. Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. The number of units after every 2 layers increase in term of power of 2, from 32 to 64, and to 128. In addition, all of them have **relu** as their activation function.

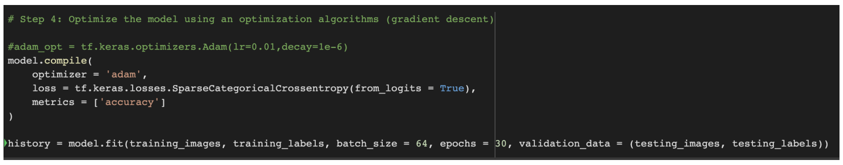
At the end of model implementation, I add **Flatten()** to flatten each image from a 2-dimensional data piece into an array, and **Dense()** with **softmax** activation function.



```
# Step 2: Implement CNN model

model = models.Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(128, activation='softmax', kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(10, activation='softmax'))

# Now display the architecture of the model
model.summary()
```

Figure 3.1: Model Implementation

Afterward, I compiled the model. To calculate the loss function, I use **Spare Categorical Cross-entropy** which measures the amount of error in statistical models. Sparse categorical cross-entropy is used when classes are mutually exclusive. This can mean that the target element of each training example may require a one-hot encoded vector with thousands of zero values, requiring significant memory. Sparse cross-entropy addresses this by performing the same cross-entropy calculation of error, without requiring that the target variable be one-hot encoded prior to training. I also use **Adam** for optimization so that my prediction can fit better and learn better to the data set. The whole process will be repeatedly implemented in **epoch = 30** times for each mini batch with **batch size = 64** to get the best result.



```
# Step 4: Optimize the model using an optimization algorithms (gradient descent)

#adam_opt = tf.keras.optimizers.Adam(lr=0.01,decay=1e-6)
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
    metrics = ['accuracy']
)

history = model.fit(training_images, training_labels, batch_size = 64, epochs = 30, validation_data = (testing_images, testing_labels))
```

Figure 3.2: Model compilation

4 Visualizing Loss and Accuracy

After building the model, I came to visualization part to check how well my model had done to classify label in each image from CIFAR-10 dataset.

The training and testing accuracy graph as well as the final testing accuracy are show below:

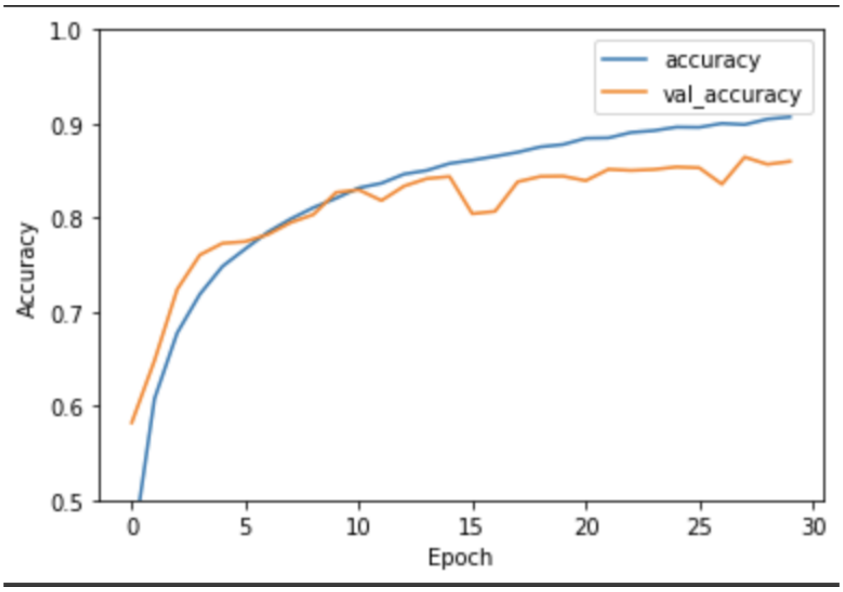
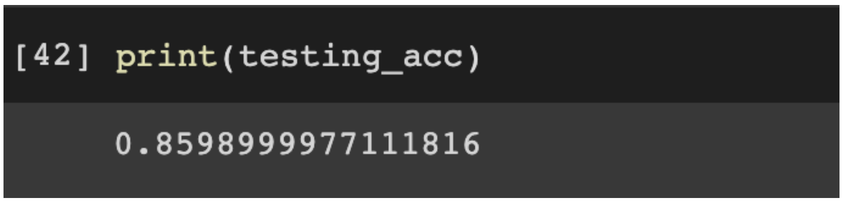


Figure 4.1: Accuracy



```
[42] print(testing_acc)

     0.8598999977111816
```

Figure 4.2: Final Accuracy

And the loss function after each epoch:

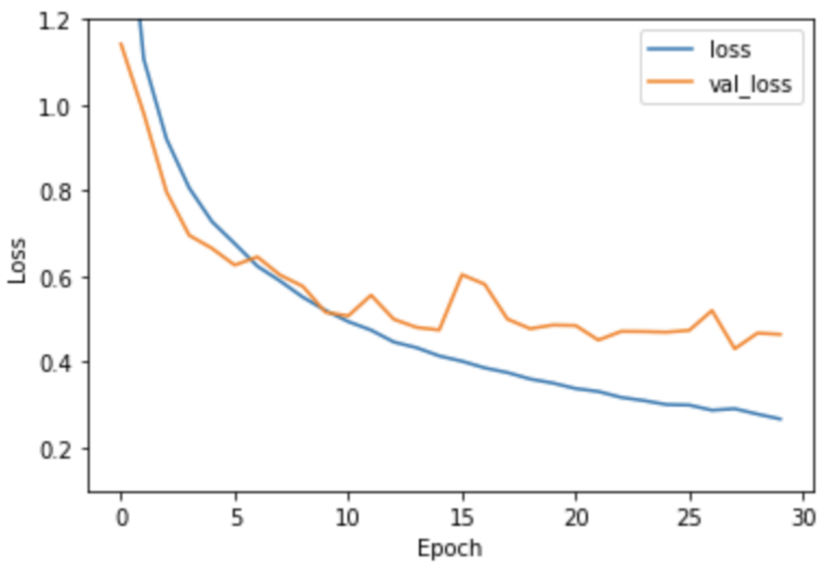


Figure 4.3: Loss function

5 Conclusion

From the given data and the corresponding graph in each case from the previous section, I can conclude that in order to make a good image classification model from a huge data set like CIFAR-10, not only using appropriate parameters and hyperparameters but what kind of model used is also very important. The number of layer units should be set to 32 initially, and then gradually increase number of units per layer to 64, and 128. The activation function should be "relu" and the epoch size should be 30. I set the epoch size to 10 previously but my model performed pretty bad testing accuracy. If I changed the epoch sixe to 50, then I would receive a much longer training time but the model performance did not improve so much. Hence, in Convolutinal Neural Network, model structure plays a very important role in traning data set, especially in with huge data set like CIFAR 10, while epoch will have a great impact on run time. Last but not least, some value which play crucial roles in fitting a good prediction like the learning rate, kind of loss function, or optimization function like Adam also need to be paid attention.